AmigaBasic: Subroutine & parametri

di Andrea de Prisco

Approfittando del fatto che questo mese la rubrica «Appunti di Informatica» ospita tematiche tutt'altro che «informatiche» come argomento Amighevole abbiamo scelto le subroutine e il passaggio dei parametri in AmigaBasic. Vedremo come si definisce una subroutine, cosa sono i parametri formali, i parametri attuali, cosa vuol dire passaggio per nome, per valore... parleremo di ambiente globale, ambiente locale statico e dinamico... tutti nomi assai familiari a chi ha già fatto un po' di informatica «seria» oltre un palmo dal proprio Basic standard.

Subroutine e dintorni

Nel (purtroppo) ben noto Basic standard, l'unico modo per utilizzare il meccanismo delle subroutine (in seguito detti anche sotto programmi) è di scrivere veri e propri pezzi di programma, terminanti con la parola chiave RETURN e saltare a questi quando serve tramite il comando GOSUB. Punto e basta. L'unico vantaggio effettivo che possiamo trarne, è un risparmio di memoria se un determinato algoritmo deve essere eseguito in più punti del nostro programma. La stessa cosa accade programmando in linguaggio macchina. Naturalmente le variabili utilizzate nel programma principale e nelle varie subroutine sono esattamente le stesse ed una modifica ad una di queste effettuata all'interno di una subroutine sarà visibile anche una volta tornati al programma principale. Ovvero non esiste il concetto di ambiente (alla spicciola: l'insieme di variabili) locale ma solo di ambiente globale.

Nei linguaggi di programmazione veramente ad alto livello, quali Pascal, Algol, Ada, Modula 2 ecc. ecc. esiste di contro una netta distinzione tra variabili «globali», definite nel programma principale, e variabili «locali» alle varie subroutine. Generalmente le variabili globali sono visibili (possono essere lette e modificate) anche all'interno dei sotto programmi; le variabili locali invece sono visibili solo all'interno della subroutine che le ha definite.

Ciò che deve essere ben chiaro è che subroutine siffatte non hanno come scopo primario il risparmio di memoria come avviene in Basic o in linguaggio macchina, ma di strutturare quanto più possibile la programmazione. Invece di scrivere un grande unico programma, definiremo tanti sotto programmi implementanti le varie funzionalità richieste, attivati da un programma principale molto corto e facilmente leggibile. A questo vantaggio già abbastanza evidente, aggiungiamo che così facendo potremo collezionare man mano una nutrita serie di subroutine che potrebbero esserci utili anche in futuro, nella realizzazione di programmi diversi.

Passaggio dei parametri

Nella definizione di un sotto programma, che come minimo conterrà un nome simbolico per poterlo invocare quando serve e naturalmente un «corpo» che descrive il da farsi, è possibile indicare una lista di parametri sui quali la subroutine opererà. Onde non confondere subito troppo le idee, facciamo un primo rapido (stupido) esempio. Immaginiamo di dover scrivere una procedura (altro sinonimo di subroutine e sotto programma) che riceve in ingresso tre variabili e copia nella prima delle tre il maggiore tra contenuto della seconda e della terza variabile. Una invocazione di tale procedura potrebbe essere ad esempio (stiamo ancora parlando in generale, non è ancora AmigaBasic):

```
CALL Max (a,b,c)
```

CALL è il comando che invoca una procedura, a, b, c le tre variabili su cui operare (Max è il nome della subroutine). I parametri passati alla procedura Max sono detti parametri attuali ovvero presenti al momento della chiamata. Nella definizione della procedura, per descrivere cosa fare con i tre parametri passati, si usano dei parametri formali, che non sono delle variabili vere e proprie, ma, ripetiamo, servono solo per indicare «primo parametro», «secondo parametro» e così via. Definiamo la nostra brava procedura:

```
PROCEDURA Max (i,j,k)
if j>k then i=j
       else i=k
end
```

Al momento della chiamata, al parametro formale «i» sarà associato il parametro attuale «a», al parametro formale «j» il parametro attuale «b», al parametro formale «k» il parametro attuale «c». L'effetto sarà quello di effettuare l'operazione descritta sulle variabili a,b,c. i, j e k non sono variabili e non hanno nulla a che vedere con eventuali omonimi presenti nel programma principale o in altri sotto programmi.

Ambiente globale e locale

L'insieme delle variabili utilizzate nel programma principale, come detto prima «per dirla molto alla spicciola», costituiscono il cosiddetto ambiente globale del programma. Se all'interno di una procedura sono utilizzate altre variabili, queste costituiranno l'ambiente locale della procedura. All'interno di una procedura saranno dunque accessibili tutte le variabili dell'ambiente locale più tutte quelle dell'ambiente globale. Nel corpo del programma principale solo quelle dell'ambiente globale. Si noti che se nel programma principale una variabile di nome A vale ad esempio 123 ed all'interno di un sotto programma viene definita (in quell'ambiente) una variabile con uguale nome, assegnato ad essa il valore 456 questo sarà il valore finché si è all'interno della procedura. Appena si esce da questa, il valore di A sarà di nuovo 123. Analogamente un'altra procedura potrà dare un nuovo valore ad A senza che cambi nulla nell'ambiente del programma principale. Si noti, non ci stancheremo mai di ripeterlo, che ciò non ha nulla a che vedere con i parametri formali appena descritti. Si parla di ambiente locale proprio per mettere in risalto il fatto che tali variabili usate dalla procedura sono utilizzate temporaneamente per l'esecuzione della subroutine stessa.

Ambiente locale statico e dinamico

Scendiamo ancora un po' in alcuni interessanti dettagli. Tutto ciò ci servirà per ben capire come funziona questa importante «fetta» di AmigaBasic.

Distinguiamo tra ambiente locale

statico e dinamico. Per essere più precisi, dato un linguaggio di programmazione in cui esiste ambiente locale per le procedure, questo è implementato in maniera statica o dinamica? La risposta è molto semplice: se l'ambiente locale è statico, chiamando più volte una stessa procedura, l'ambiente si conserverà tra una chiamata e l'altra (procedure «con stato interno»). Se ogni volta si annulla, ovvero perdiamo il contenuto di tutte le variabili locali, si tratta di ambiente locale dinamico. Vantaggi e svantaggi possono essere ovvi o meno. Con l'ambiente locale dinamico, non occupiamo spazio di memoria per conservare l'ambiente, ma tale spazio viene allocato al momento della chiamata e deallocato al ritorno al programma principale. Capirete che se esistono molte procedure, nel caso statico, dobbiamo allocare molto spazio di memoria, senza contare che magari molte di queste non saranno più chiamate e quindi conservare le variabili può essere in tal senso costoso. Altro vantaggio a favore dell'ambiente locale dinamico, la possibilità che una procedura richiami se stessa (ricorsione).

Se però siamo interessati a procedure con stato interno, non possiamo che sperare che il nostro linguaggio di programmazione implementi l'ambiente locale in modo statico. Con l'ambiente locale dinamico, per realizzare la stessa cosa, dovremmo ogni volta passare lo stato interno al programma principale che ce lo restituirà alla prossima chiamata. Come dire... macchinoso ma possibile.

Ancora sui parametri

L'ultimo problema da risolvere prima di passare all'AmigaBasic riguarda il passaggio dei parametri. Ovvero cosa effettivamente passiamo alla procedura al momento della chiamata e cosa questa ci restituirà al ritorno. Potremmo, ad esempio, passare direttamente valori o variabili. Ma come si sa, una variabile denota un valore quindi, di primo acchito potrebbe sembrare la stessa cosa. Nell'esempio visto precedentemente, la procedura Max di cui sopra, i parametri attuali erano tre variabili, a, b, c, ma avremmo potuto chiamare la subroutine anche passando una variabile e due valori, ad esempio:

```
CALL Max(a,3,4)
```

nel qual caso in «a» sarebbe finito il 4. Sicuramente non avremmo potuto passare tre numeri: che senso avrebbe avuto l'assegnamento di un numero ad un altro? Emerge a questo punto un'altra importante distinzione tra passare alla procedura le variabili o i valori denotati da queste. Per capire meglio, facciamo un ulteriore esempio: facciamo una modifica alla procedura Max:

```
PROCEDURA Max (i,j,k)
j=j+4
if j>k then i=j
       else i=k
end
```

Max così modificata, supposto di effettuare una chiamata di questo tipo:

```
CALL Max(a,b,c)
```

restituirà nella prima variabile passata il numero più grande tra il contenuto della seconda variabile aumentato di 4 e il contenuto della terza variabile. Questo sia che il passaggio del secondo e terzo parametro era inteso per valore (è passato alla procedura il valore della variabile in quel momento), o per «nome» (nel corpo della procedura noi manipoliamo effettivamente le variabili ricevute). La differenza tra i due modi di passaggio dei parametri sta nel fatto che nel primo caso, in seguito all'istruzione j=j+4 non modifichiamo anche la b del programma chiamante, nel secondo caso l'incremento di 4 lo effettuiamo sia nell'ambiente locale (valore di j) che in quello globale (valore di b). Complicato?

AmigaBasic

Nell'AmigaBasic l'ambiente locale delle procedure è gestito staticamente, il passaggio dei parametri può avvenire sia per nome che per valore, è possibile esportare variabili dall'ambiente locale a quello globale, mentre è obbligatorio importare esplicitamente le variabili dell'ambiente globale che vogliamo «vedere» all'interno di una procedura. Per prima cosa vediamo come si definisce una subroutine. La sintassi è la seguente:

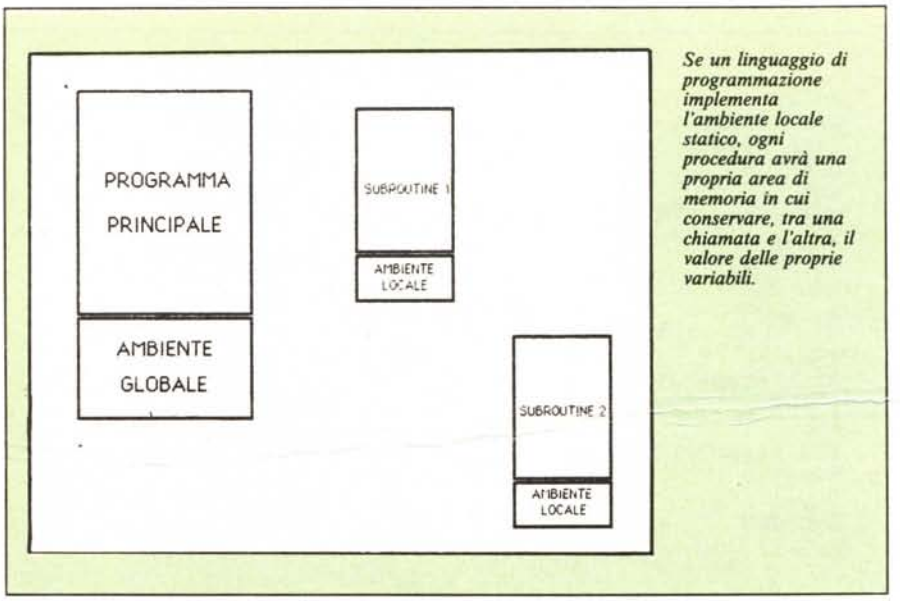
```
SUB NomeProcedura(parametri) STATIC
corpo della procedura
END SUB
```

La prima parola chiave, SUB, indica che stiamo dichiarando una procedura. Segue il nome di questa e, tra parentesi, i parametri adoperati. È obbligatoria la chiave STATIC, che sta ad indicare (e non se ne può fare a meno) che l'ambiente locale è gestito staticamente. In AmigaBasic, per chiamare una procedura basta invocare il suo nome, facoltativamente preceduto dalla parola chiave CALL.

Nel corpo della procedura, ogni riferimento a variabili sarà inteso locale, quindi se adoperiamo ad esempio la variabile X, questa sarà creata nell'ambiente locale anche se ne esiste una omonima nel corpo del programma principale. Se, di contro, vogliamo adoperare variabili globali useremo il comando SHARED per importarle. Lo stesso comando serve per esportare variabili da locali a globali.

Immaginiamo che nel programma principale esista una variabile T che intendiamo utilizzare all'interno della procedura. Scriveremo:

```
SUB Pippo STATIC
SHARED T
.
.
.
END SUB
```



Se un linguaggio di programmazione implementa l'ambiente locale statico, ogni procedura avrà una propria area di memoria in cui conservare, tra una chiamata e l'altra, il valore delle proprie variabili.

In figura 1 troviamo un banale esempio di importazione di variabile. Nel programma principale la variabile X vale 3, con la chiamata CALL Pippo che invoca l'esecuzione della procedura, vedremo tale valore stampato sul video. Se omettevamo lo SHARED X nel corpo di Pippo, il valore stampato sarebbe stato naturalmente zero dato che si sarebbe trattato di una nuova variabile locale, ben diversa, anche se con uguale nome, da quella del programma principale.

Per esportare variabili locali, ovvero per renderle globali una volta utilizzate all'interno della procedura, come detto si utilizza ugualmente il comando SHARED. Basta indicare di seguito a questo il nome della variabile (naturalmente non già esistente) che intendiamo rendere visibile anche al programma principale dopo l'esecuzione della procedura. Semplice, no?

```
x=3
CALL pippo
END
SUB pippo STATIC
SHARED x
PRINT x
END SUB
```
Figura 1

Ambiente locale statico

In figura 2 è mostrato una prova del fatto che esiste un vero e proprio ambiente locale e che questo è gestito staticamente. Il programma principale non fa altro che assegnare alla sua variabile A il valore 100, chiamare due volte la subroutine Pippo, infine stampare il valore della A. Ad ogni chiamata di Pippo, si incrementa il valore della A e si stampa il contenuto di questa. Il risultato è dunque di vedere stampati tre valori sul nostro video. Quali saranno questi tre valori? Semplice: 1,2,100. Il motivo è ovvio: all'interno di Pippo la A è una nuova variabile quindi all'inizio vale zero, se l'incrementiamo varrà 1 (primo valore stampato). Alla seconda chiamata, essendo l'ambiente locale statico, non perdiamo il contenuto della variabile locale A quindi col nuovo incremento la portiamo a 2 (secondo valore stampato). Infine, tornati per la seconda volta al programma principale, stampiamo il contenuto della A globale che vale (ed è sempre valsa) 100.

Passaggio dei parametri

Nella definizione di una subroutine, come detto, possiamo indicare una lista di parametri su cui lavorerà la procedura. Dovremo indicare il tipo dei parametri tramite le solite convenzioni del Basic e se si tratta di array, dovremo indicare il numero di dimensioni di questo (non la sua dimensione). Ad esempio, una dichiarazione del tipo:

```
SUB Tip(A%,B,C$,D(2)) STATIC
END SUB
```

dichiara una procedura di nome Tip il cui primo parametro è un intero, il secondo un floating point (ovvero non intero), il terzo una stringa, il quarto un array bidimensionale (una matrice). Come più volte ripetuto i nomi di questi sono formali e servono solo per riferire, all'interno della procedura, i parametri attuali che passeremo al momento della chiamata. Da ricordare sempre, che l'AmigaBasic non esegue la «coercion» dei tipi quindi se abbiamo detto che il secondo parametro è un intero, intero dovrà essere... pena proROMPENTE messaggio d'errore. Se ad esempio dobbiamo passare un numero, e la procedura aveva un parametro di tipo floating point, non potremo scrivere Pippo(3) ma saremo costretti a scrivere Pippo(3.0). Per la cronaca, appuntate anche che le espressioni dell'AmigaBasic sono dello stesso tipo dei suoi operandi: 3+1 è un'espressione intera, 3+1.0 è un'espressione floating point (e qui alla Microsoft ci hanno graziati: almeno per gli operatori la coercion avviene!).

Per nome o per valore?

È, fortunatamente, possibile in tutti e due i modi. Al momento della dichiarazione non cambia nulla (quindi diciamo che è comunque per nome), ma al momento della chiamata possiamo barare facendo diventare il passaggio di una variabile per valore. Come?, semplice: se racchiudiamo tra parentesi la variabile, passeremo soltanto il valore da essa denotato quindi modifiche al parametro, all'interno della procedura, non si ripercuoteranno nell'ambiente globale.

In figura 3 è mostrato un primo esempio. Il programma principale assegna alla variabile X il valore 3. Immediatamente dopo chiama la subroutine Pippo passando come parametro la X, per poi stampare il contenuto di questa. All'interno della procedura semplicemente stampiamo il valore del parametro ricevuto e provvediamo ad incrementarlo. Dato che il parametro così passato è per nome, l'incremento appena effettuato si ripercuoterà anche sulla X dell'ambiente globale dunque il risultato sarà di veder stampato prima un 3 e poi un bel 4.

In figura 4 abbiamo lo stesso programma, ma il passaggio del parametro questa volta avviene per valore. Si notino, all'uopo, le parentesi che racchiudono la X nella chiamata di Pippo. Questa volta l'incremento non riguarderà il parametro attuale (X) dunque ritornando al programma principale la X varrà ancora 3. Il risultato di tutto sarà la stampa di due 3.

Infine, in figura 5, abbiamo passato come parametro una espressione (quindi per valore... per forza) e la stampa della X la effettuiamo all'interno della procedura. Vediamo se siete diventati bravi: prima di dare RUN, indovinate cosa stamperà il programma. Coraggio, non è difficile...

```
a=100
CALL pippo
CALL pippo
PRINT a
END
SUB pippo STATIC
a=a+1
PRINT a
END SUB
```
Figura 2

```
x=3
CALL pippo(x)
PRINT x
END
SUB pippo(y) STATIC
PRINT y
y=y+1
END SUB
```
Figura 3

```
x=3
CALL pippo((x))
PRINT x
END
SUB pippo(y) STATIC
PRINT y
y=y+1
END SUB
```
Figura 4

```
x=3
CALL pippo(x*4+3)
END
SUB pippo(y) STATIC
PRINT y
y=y+1
PRINT x,y
END SUB
```
Figura 5